Quantitative Anonymity Evaluation of Voting Protocols

Fabrizio Biondi, Axel Legay

Quantitative Anonymity Evaluation of Voting Protocols

Fabrizio Biondi[1] and Axel Legay[1]

INRIA Rennes, France

Abstract. In an election, it is imperative that the vote of the single voters remain anonymous and undisclosed. Alas, modern anonymity approaches acknowledge that there is an unavoidable leak of anonymity just by publishing data related to the secret, like the election's result. Information theory is applied to quantify this leak and ascertain that it remains below an acceptable threshold.
We apply modern quantitative anonymity analysis techniques via the state-of-the-art QUAIL tool to the voting scenario. We consider different voting typologies and establish which are more effective in protecting the voter's privacy. We further demonstrate the effectiveness of the protocols in protecting the privacy of the single voters, deriving an important desirable property of protocols depending on composite secrets.

1 Introduction

Voting is the backbone of the democratic process [15]. To be effective, a voting system must allow the voters to freely express their opinion and elect the public officials that will represent them in the government. An effective voting system guarantees that each vote is counted exactly once, that no malicious agent can tamper with the results of the vote, and that no vote can be traced back to the voter who cast it.

Various traditional and electronic voting systems have been proposed to assure such guarantees. The use of cryptography and certification authorities can guarantee that only eligible voters can vote and that their vote is counted exactly once, and the production of fake credentials can safeguard voters against being coerced to reveal their vote [12]. The anonymity of the vote is harder to guarantee; current proposals include assumptions on the absolute anonymity of the voting channels [12,14] or expect enterprises and universities to provide public proxy servers to hide the IP address of the voter [10]. The problem with these approaches to anonymity is that the anonymity of a vote is considered a qualitative, yes/no property, verifying whether it is possible for an attacker to infer any amount of information about the identity of the voters; this is known as the *possibilistic* approach. The *probabilistic* approach instead considers anonymity as a quantity that can be decreased by the attack of an external agent and by other factor in the voting process including the magnitude of the electoral seat, the electoral formula used, the results of the elections and the number of candidates. Since none of these factors completely compromise anonymity, a qualitative technique has to either ignore them or consider any voting protocol unsafe.

Current approaches to anonymity consider a secret, like the identity of the caster of a vote, as a quantitative amount of information, and use information theory to quantify

how much of this secret information is inferred by a malicious attacker [8,9]. This amount is called *information leakage*. The qualitative approach tags as insecure even a negligible amount of loss of information, in practice considering any real system insecure except under very strong assumptions. On the other hand, the quantitative approach allows the analyzer to determine a bound above which a loss of anonymity is considered noteworthy. Quantitative anonymity analysis has been applied among others to study the trade-off between anonymity and utility of operations on databases [2] and to define information-theoretical bounds to differential privacy [4]. To the best of our knowledge, these techniques have not been applied to voting protocols.

Completely automated tools have been created to quantify information leakage for any secret-dependent protocol. Previously we have introduced a theoretical framework to model protocols with Markov chains and efficiently and precisely compute their leakage [5]. We implemented the approach in the QUAIL tool, the first tool able to perform an arbitrary-precision leakage analysis of a non-deterministic secret-dependent protocol [7].

In this work we will use QUAIL to analyze different typologies of electoral formulae. QUAIL considers the combined votes of the voters as a precise amount of secret bits and quantifies precisely how many of these bits are inferred by an attacker able to read the published results of the elections, i.e. the information leakage. Since these results are public, there is no way to avoid this loss of anonymity. Consequently any qualitative method that claims to perfectly guarantee anonymity of the voters is ignoring this non-negative information leakage. By quantifying exactly this amount we establish a lower bound on the amount of anonymity that any implementation or formula can guarantee.

We study two very general typologies of electoral formulae: Single Preference formulae, where each voter expresses a preference for a candidate, and Preference Ranking formulae, where each voter ranks all candidates from the best to the worst. Our results are valid for any electoral formula in the typologies. This classification is traversal to the common division in proportional and majoritarian systems, as it depends only on the way the vote is expressed.

We consider that the secret is not a single entity, but a composite of the secret votes. If we have 10 secrets and each is 2 bits and the leakage of the system is 2 bits, we need to identify whether the leaked information corresponds to the secret of one of the voters or only to general information about the result of the vote. The theoretical bases for the study of leakage of composite secrets are very recent [3], and to the best of our knowledge no large scale study of practical cases has been published.

Our analysis shows the exact impact of the magnitude of the electoral seat and the number of candidates on the anonymity of the vote. The results suggest the data from seats of low magnitude, like hospital seats, should be aggregated before publication to protect the voters' anonymity. Also we show that the leakage on a single voter's secret is strictly less than the leakage on all secrets divided by the number of voters, proving that the voting protocols analyzed are effective in protecting the single voters' secrets. This is a generally desirable property for protocols on composite secrets.

The rest of the paper is organized as follows: Section 2 introduces common concepts in probability and information theory and Section 3 details the leakage theory and how it is implemented in the QUAIL analyzer. Section 4 presents the two typologies of electoral

formulae we analyze and in Section 5 we discuss the results obtained. Section 6 explains which problems we are facing in obtaining more results and what steps we are taking to solve them.

2 Background

We define common concepts in probability theory and information theory that are used through the paper. We refer to books on the subject [13] for the basic definitions on probability theory. We call X a *discrete random variable* and $\mathcal{X}$ a *discrete stochastic process*, i.e. an indexed infinite sequence of discrete random variables $(X_0.X_1, X_2, ...)$ ranging over the same sample space S. The index of the random variables in a stochastic process can be understood as modeling a concept of discrete time, so X_k is the random variable representing the system at time unit k.

2.1 Markov Chains

A discrete stochastic process is a *Markov chain* $\mathcal{C} = (C_0, C_1, C_2, ...)$ iff $\forall k \in \mathbb{N}$. $P(C_k|C_{k-1}, ..., C_0) = P(C_k|C_{k-1})$. A Markov chain on a sample space S can also be defined as follows:

Definition 1. *A tuple $\mathcal{C} = (S, s_0, P)$ is a Markov Chain (MC), if S is a finite set of states, $s_0 \in S$ is the initial state and P is an $|S| \times |S|$ probability transition matrix, so $\forall s, t \in S$. $P_{s,t} \geq 0$ and $\forall s \in S$. $\sum_{t \in S} P_{s,t} = 1$.*

We call $\pi^{(k)}$ the probability distribution vector over S at time k and $\pi_s^{(k)}$ the probability $\pi^k(s)$ of visiting the state s at time k. This means that, considering a Markov chain $\mathcal{C}$ as a time-indexed discrete stochastic process $(C_0, C_1, ...)$, we write $\pi^{(k)}$ for the probability distribution over the random variable C_k. Since we assume that the chain starts in state s_0, then $\pi_s^{(0)}$ is 1 if $s = s_0$ and 0 otherwise. Note that $\pi^{(k)} = \pi_0 P^k$, where P^k is matrix P elevated to power k, and P^0 is the identity matrix of size $|S| \times |S|$.

A state $s \in S$ is *absorbing* if $P_{s,s} = 1$. In the figures we will not draw the looping transition of the absorbing states, to reduce clutter. We say that a Markov chain is *one-step* if all states except the starting state s_0 are absorbing. We will usually refer to one-step Markov chains as $\mathbb{C}$.

Let $\xi(s, t)$ denote the *expected residence time* in a state t in an execution starting from state s given by $\xi(s, t) = \sum_{n=0}^{\infty} P_{s,t}^n$. We will write ξ_s for $\xi(s_0, s)$.

We will enrich our Markov chains with a finite set $\mathtt{V}$ of natural-valued variables, and for simplicity we assume that there is a very large finite bit-size M such that a variable is at most M bit long. We define an assignment function $A : S \to [0, 2^M - 1]^{|\mathtt{V}|}$ assigning to each state the values of the variables in that state. We will use the expression $\mathtt{v}(s)$ to denote the value of the variable $\mathtt{v} \in \mathtt{V}$ in the state $s \in S$.

Given a Markov chain $\mathcal{C} = (S, s_0, P)$ let a *discrimination relation* $\mathcal{R}$ be an equivalence relation over S. We use discrimination relation to quotient one-step Markov chains:

Definition 2. *Given a one-step Markov chain $\mathbb{C} = (S, s_0, P)$ and a discrimination relation $\mathcal{R}$ over S, we define the* quotient $\mathbb{C}/\mathcal{R}$ *of $\mathbb{C}$ over $\mathcal{R}$ as the one-step Markov chain $\mathbb{C}/\mathcal{R} = (\bar{S}, \bar{s}_0, \bar{P})$ where*

- *$\bar{S}$ is the set of equivalence classes of S induced by $\mathcal{R}$;*
- *$\bar{s}_0$ is the equivalence class of s_0;*
- *for each equivalence class $\bar{s} \in S'$, $\bar{P}_{\bar{s}_0,\bar{s}} = \sum_{s \in \bar{s}} P_{s_0,s}$ and $\bar{P}_{\bar{s},\bar{s}} = 1$.*

2.2 Information Theory

The *entropy* of a probability distribution is a measure of the unpredictability of the events considered in the distribution [17].

Definition 3. *[13] Let X and Y be two random variables with probability mass functions $p(x)$ and $p(y)$ respectively and joint pmf $p(x, y)$. Then we define the following non-negative real-valued functions:*

- *Entropy* $H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$
- *Joint entropy* $H(X, Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(x, y)$
- *Conditional entropy* $H(X|Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(x|y) =$
  $= \sum_{y \in Y} p(y) H(X|Y = y) = \sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log_2 p(x|y) =$
  $= H(X, Y) - H(Y)$ *(chain rule)*
- *Mutual information* $I(X; Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 \left(\frac{p(x,y)}{p(x)p(y)} \right) =$
  $= H(X) + H(Y) - H(X, Y) \leq \min(H(X), H(Y))$

Since every state s in a MC $\mathcal{C}$ has a discrete probability distribution over the successor states we can calculate the entropy of this distribution. We will call it *local entropy*, $L(s)$, of s: $L(s) = -\sum_{t \in S} P_{s,t} \log_2 P_{s,t}$. Note that $L(s) \leq \log_2(|S|)$.

As a MC $\mathcal{C}$ can be seen as a discrete probability distribution over all of its possible traces, we can assign a single entropy value $H(\mathcal{C})$ to it. The global entropy $H(\mathcal{C})$ of $\mathcal{C}$ can be computed by considering the local entropy $L(s)$ as the expected reward of a state s and then computing the expected total reward of the chain [6]: $H(\mathcal{C}) = \sum_{s \in S} L(s) \xi_s$ If a Markov chain is one-step its entropy corresponds to the local entropy of the initial state s_0.

3 Information Leakage of Markov Chains

3.1 Theoretical background

We use information theory to compute the amount of bits of a secret variable $\mathtt{h}$ that can be inferred by an attacker able to observe the value of an observable variable $\mathtt{o}$ after the termination of a protocol. We call this amount *Shannon leakage* or just *leakage*, and it corresponds to the mutual information between the distribution on the secret and the distribution on the observable variable. This analysis assumes the worst possible attacker: the attacker has access to the source code of the protocol and to unlimited computational power.

We will model system-attacker scenarios with Markov chains in which to each state we associate a unique assignment of values to all variables [5]. Then we define leakage as follows:

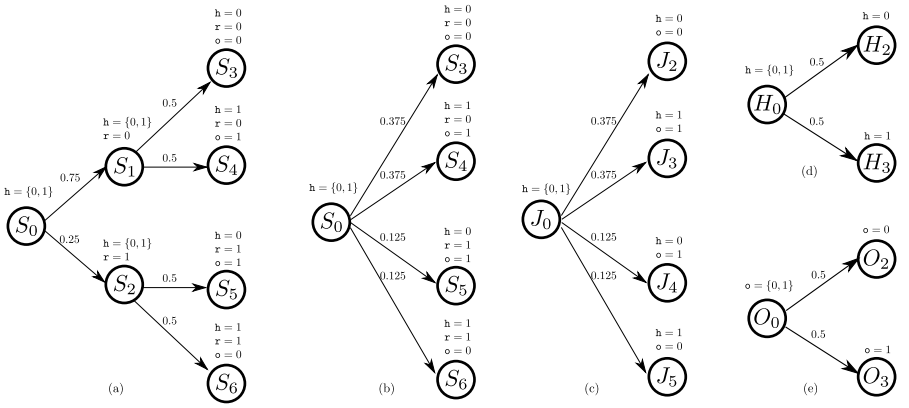

Fig. 1: Bit XOR example: a) Markov chain semantics $\mathcal{C}$. b) Observable reduction $\mathbb{C}$. c) Joint quotient $\mathbb{C}_{|(o,h)}$. d) Secret's quotient $\mathbb{C}_{|h}$. e) Observer's quotient $\mathbb{C}_{|o}$.

Definition 4. *Let $\mathcal{C}$ be a Markov chain enriched with variable from the set* V. *Let* h *represent the secret variables and* O *the variables whose value is observable to the attacker. Then we define the* Shannon leakage *of $\mathcal{C}$ as the mutual information $I(\mathtt{O};\mathtt{h})$ between the secret and observable variables.*

Note that to compute leakage we need to have a prior probability distribution over the secret, modeling what the attacker knows before observing the observable output of the protocol. We will assume for simplicity that the attacker knows the possible values of the secret, since he can read the source code and verify which kind of variable holds it, but has no additional information about it.

The modeling of a system-attacker scenario as a Markov chain starts by dividing the system's variables in private and public variables. Private variables, including the secret variable h, are the ones whose value is not defined at compilation time. In each state of the Markov chain a set of allowed values is assigned to each private variable. Public variables, including the observable variable o and the program counter pc, are variable whose value is known to the analyst. On each state a given value is assigned to each public variable.

Given the source code of the system and a prior distribution over the private variables, we have enough information to build the Markov chain semantics $\mathcal{C}$ of the protocol, since for each state we can determine its successor states and the corresponding transition probabilities.

We show a simple example, and refer to [5] for the complete semantics. Let secret variable h be a secret bit, observable variable o an observable bit and public variable r a random bit being assigned the value 0 with probability 0.75 and 1 otherwise. We assign to o the result of the exclusive OR between h and r and terminate. We want to quantify the amount of information about h that can be inferred by knowing the value of o. The Markov chain semantics $\mathcal{C}$ for the example is shown in Fig. 1a. Each state is enriched with information about the allowed values of private variables and the values of public variables, e.g. in state S_1 secret variable h can be either 0 or 1 and public variable r is 0.

Subsequently we need to model the fact that the attacker has to wait for the protocol to terminate to read the observable output. Equivalently, we can say that the attacker is not aware of the internal behavior of the system; this is modeled by hiding in the Markov chain model the internal states, i.e. all states except the initial state and the absorbing states. We call the resulting Markov chain the *observable reduction* $\mathbb{C}$. The observable reduction for the example is shown in Fig. 1b.

Note that the observable reduction is a one-step Markov chain, so we can compute quotients on it following Definition 2. To compute the leakage we need to compute three quotients from the observable reduction:

Joint quotient The joint quotient process $\mathbb{C}_{|(o,h)}$ models the joint behavior of the secret and observable variables. It is shown in Fig. 1c.

Secret's quotient The secret's quotient process $\mathbb{C}_{|h}$ models the behavior of the secret variable. It is shown in Fig. 1d.

Observer's quotient The observer's quotient process $\mathbb{C}_{|o}$ models the behavior of the observable variable. It is shown in Fig. 1e.

Finally we compute leakage as the mutual information $I(\mathtt{O};\mathtt{h})$, as between the secret and observable variable, as explained in Definition 4. To compute it we apply the formula $I(X;Y) = H(X) + H(Y) - H(X,Y)$ from Definition 3, obtaining

$$I(\mathtt{o};\mathtt{h}) = I(\mathbb{C}_{|o};\mathbb{C}_{|h}) = H(\mathbb{C}_{|o}) + H(\mathbb{C}_{|h}) - H(\mathbb{C}_{|(o,h)})$$

meaning that the leakage can be computed as the sum of the entropies of the secret and observable quotient minus the entropy of the joint quotient. In our example we have $(\mathbb{C}_{|o}) = 1$, $H(\mathbb{C}_{|h}) = 1$ and $H(\mathbb{C}_{|(o,h)}) = 1.8112...$, so we conclude that the program leaks $1 + 1 - 1.8112... \approx 0.1887$ bits, or 18.87% of the secret.

3.2 QUAIL Implementation

The QUAIL tool quantifies the Shannon leakage of a probabilistic process in a fully automated way. The user just provides the source code for the process in the QUAIL imperative language, specifying the size of the variables and whether each variable is public, private, secret or observable. QUAIL computes the expected amount of information about the secret variables that an attacker is able to infer by knowing the values of the observable variables after the protocol has terminated and produced output, implementing the theory presented in Section 3.1.

The QUAIL language is a WHILE language enriched with `for` loops, multidimensional arrays and constant declarations. For simplicity all variables are integer variables; the bit length of each variable is defined by the coder at declaration time. Also, assignment to private or secret variables is not allowed, and all variables have to be declared at the beginning of the program. We refer to [1] for the source code and full semantics of QUAIL.

QUAIL is able to handle secret and private variables with a large number of possible values. Alas, the representation of arbitrary probability distributions on such variables on a real machine is untreatable: a probability distribution over a 64-bit variable is composed of $2^{64} \approx 1/8 \cdot 10^{19}$ rational numbers. For this reason QUAIL does not allow

the user to define arbitrary prior distributions over the secret, and always assumes that the prior distribution is uniform.

```
1 secret int1 h;
2 observable int1 o;
3 public int1 r;
4 random r := randombit
      (0.75);
5 assign o := h ^ r;
6 return;
```

Fig. 2: Bit XOR example: source code.

The QUAIL source code for the bit XOR example presented in Section 3.1 is shown in Fig. 2. In lines 1 to 3 the variables are declared, then in line 4 variable `r` is assigned value 0 with probability 0.75 and 1 otherwise, and in line 5 variable `o` is assigned with the exclusive OR of variables `h` and `r`.

We save the source code in Fig. 2 in a file `bitxor.quail` and invoke QUAIL with the command

```
./quail bitxor.quail -v 0 -p 5
```

where `-v 0` suppresses all output except for the leakage result and `-p 5` specifies that we want 5 significant digits in the answer. QUAIL outputs

```
0.18872
```

showing that the program leaks ≈ 0.18872 bits of the secret, in accordance to what we computed theoretically in Section 3.1.

4 Modeling Voting Protocols

In an election, each voter is called to express his preference for the competing candidates. The *voting system* defines the way the voters express their preference: either on paper in a traditional election, or electronically in e-voting. The voting system also comprehends the additional procedures enforced to guarantee that the voters can vote freely, that they can verify that their vote has been counted and that their vote remains confidential.

After the votes have been cast, the *results* of the vote are published, usually in an aggregated form to protect the anonymity of the voters. Finally, the winning candidate or candidates is chosen according to a given *electoral formula*.

In this section we present two different typologies of voting, representing two different ways in which the voters can express their preference: in the *Single Preference* protocol the voters declare their preference for exactly one of the candidates, while in the *Preference Ranking* protocol each voter ranks the candidate from his most favorite to his least favorite. Since each protocol we model is concerned only about how the votes are expressed and counted and what results are published, each protocol models a number of electoral formulae. For the same reason, the models are valid both for uninominal and multinominal elections.

4.1 Single Preference

The Single Preference protocol typology models all electoral formulae in which each of the N voters expresses one vote for one of the C candidates, including plurality and majority voting systems and single non-transferable vote [15]. The votes for each candidate are summed up and only the results are published, thus hiding information

about which voter voted for which candidate. The candidate or candidates to be elected are decided according to the electoral formula used.

Secret and observable encoding. The secrets and observables are modeled by the following lines of QUAIL code:

```
secret array [N] of int32 vote:=[0,C-1];
observable array [C] of int32 result;
```

The secret is an array of integers with a value for each of the N voters. Each value is a number from 0 to C-1, representing a vote for one of the C candidates. The observable is an array of integers with the votes obtained by each of the C candidates. The full model for this protocol is shown in the Appendix due to space constraints.

Formal leakage computation. The protocol is simple, and its information leakage can be computed formally, as shown by the following lemma:

Lemma 1. *The information leakage for the Single Preference protocol with n voters and c candidates corresponds to*

$$\log_2 c^n - \frac{1}{c^n} \sum_{k_1+k_2+\ldots+k_c=n} \binom{n}{k_1+k_2+\ldots+k_c} \left(\log_2 \binom{n}{k_1+k_2+\ldots+k_c}\right)$$

The proof for Lemma 1 is in the Appendix. While the lemma characterizes the solution computed by QUAIL for this case, it is very hard to find such a characterization for any process, so in general QUAIL is the best way to obtain a result. We run QUAIL with the command

```
./quail single_preference.quail -v 0 -p 5
```

with the same parameters we used in Section 3.2 and obtain

```
1.8112
```

showing that the leakage of the Single Preference protocol for 3 voters and 2 candidates is ≈ 1.8112 bits.

4.2 Preference Ranking

The Preference Ranking protocol typology models all electoral formulae in which each of the n voters expresses an order of preference of the c candidates, including the alternative vote and single transferable vote systems [15]. In the Preferential Voting protocol the voter does not express a single vote, but rather a ranking of the candidates; thus if the candidates are A, B, C and D the voter could express the fact that he prefers B, then D, then C and finally A. Then each candidate gets c points for each time he appears as first choice, $c-1$ points for each time he appears as second choice, and so on. The points of each candidate are summed up and the results are published.

Secret and observable encoding. The secrets and observables are modeled by the following lines of QUAIL code:

```
secret array [N] of int32 vote:=[0,C!-1];
observable array [C] of int32 result;
```

The secret is an array of integers with a value for each of the N voters. Each value is a number from 0 to C!-1, representing one of the possible C! rankings of the C candidates. The observable is an array of integers with the points obtained by each of the C candidates. The full model for this protocol is shown in the Appendix due to space constraints.

5 Experimental Results

We discuss some of the initial results we have obtained by analyzing the Single Preference and Preference Ranking voting protocols.

Single	**Voters**		
	2	**3**	**4**
Cands **2**	1.50	1.81	2.03
3	/	3.12	3.57
4	/	/	4.81

Ranking	**Voters**		
	2	**3**	**4**
Cands **2**	1.50	1.81	2.03
3	/	2.54	2.96
4	/	/	timeout

Table 1: Voting protocols: leakage tables for Single Preference (on the left) and Preference Ranking (on the right)

Leakage comparison. Table 1 shows the leakage amounts for the Single Preference and Preference Ranking protocols for different numbers of voters and candidates. We note that the results for 2 candidates are identical, since in this case in both protocols the voters can vote in only 2 different ways. The table shows that the leakage for the Preference Ranking protocol is in general lower than the leakage for the Single Preference protocol. Nonetheless we are comparing protocols with a different secret size, so it is more appropriate to compare posterior entropies.

Single	**Voters**		
	2	**3**	**4**
Cands **2**	0.50	1.19	1.97
3	/	1.63	2.76
4	/	/	3.19

Ranking	**Voters**		
	2	**3**	**4**
Cands **2**	0.5	1.19	1.97
3	/	5.21	7.37
4	/	/	timeout

Table 2: Voting protocols: posterior entropy tables for Single Preference (on the left) and Preference Ranking (on the right)

Posterior entropy comparison. In Table 2 we show the posterior entropies for the same cases as Table 1. The result confirm that the protocols are identical in case there are 2 candidates, and Preference Ranking is more efficient in protecting the anonymity of the votes than Single Preference.

3V-2C	H_O	H_h	$H_{O,h}$	$I_{O,h}$	$I_{O,h}\%$	pH_h
1	1.81	1.00	2.50	0.31	31.1%	0.69
2	1.81	2.00	3.00	0.81	40.5%	1.19
3	1.81	3.00	3.00	1.81	60.3%	1.19

3V-3C	H_O	H_h	$H_{O,h}$	$I_{O,h}$	$I_{O,h}\%$	pH_h
1	3.12	1.58	4.08	0.62	39.1%	0.96
2	3.12	3.16	4.75	1.53	48.5%	1.63
3	3.12	4.75	4.75	3.12	65.6%	1.63

4V-2C	H_O	H_h	$H_{O,h}$	$I_{O,h}$	$I_{O,h}\%$	pH_h
1	2.03	1.00	2.81	0.21	21.0%	0.79
2	2.03	2.00	3.50	0.53	26.5%	1.47
3	2.03	3.00	4.00	1.03	34.0%	1.97
4	2.03	4.00	4.00	2.03	50.0%	1.97

4V-3C	H_O	H_h	$H_{O,h}$	$I_{O,h}$	$I_{O,h}\%$	pH_h
1	3.57	1.58	4.70	0.45	28.7%	1.13
2	3.57	3.16	5.67	1.07	34.0%	2.09
3	3.57	4.75	6.33	1.99	41.9%	2.76
4	3.57	6.33	6.33	3.57	56.3%	2.76

5V-2C	H_O	H_h	$H_{O,h}$	$I_{O,h}$	$I_{O,h}\%$	pH_h
1	2.19	1.00	3.03	0.16	16.7%	0.84
2	2.19	2.00	3.81	0.38	19.3%	1.62
3	2.19	3.00	4.50	0.69	23.2%	2.31
4	2.19	4.00	5.00	1.19	30.0%	2.81
5	2.19	5.00	5.00	2.19	44.0%	2.81

5V-3C	H_O	H_h	$H_{O,h}$	$I_{O,h}$	$I_{O,h}\%$	pH_h
1	3.93	1.58	5.16	0.35	22.3%	1.23
2	3.93	3.16	6.29	0.80	25.5%	2.36
3	3.93	4.75	7.25	1.43	30.1%	3.32
4	3.93	6.33	7.92	2.34	37.0%	3.99
5	3.93	7.92	7.92	3.93	49.6%	3.99

Table 3: Single Preference voting protocol: entropies and leakage on varying the number of voters, candidates and target voters.

Analysis of Single Preference with variable number of targets. In Table 3 we give detailed results of the analysis of the Single Preference voting protocol, on varying the number of voters, candidates and target voters. The code in bold on the top left corner of a table shows how many voters and candidates are being considered in the experiment, e.g. **4V-2C** means 4 voters and 2 candidates. The column of the left represents the number of target voters for the experiment, i.e. how many votes is the attacker trying to infer. The table reports the following values:

H_O is the entropy of the observer's quotient
H_h is the (prior) entropy of the secret's quotient
$H_{O,h}$ is the entropy of the joint quotient
$I_{O,h}$ = $H_O + H_h - H_{O,h}$ is the information leakage
$I_{O,h}\%$ = $I_{O,h}/H_h$ is the percentage of the secret that has been leaked
pH_h = $H_h - I_{O,h}$ is the (expected) posterior entropy of the secret, i.e. the amount of secret that has not been leaked

Discussion. Since the posterior entropy measures how hard it would be for an attacker to learn the secret after observing the results of the voting, we focus on it as the measure of how confidential the votes are after the attack. Note that posterior entropy increases less than linearly with the number of targets, so if we want to learn both the votes of voters Alice and Bob it is more convenient to consider the two votes as a single composite secret than to try to learn the two votes separately. Note also that the posterior entropy when all voters are targets is the same as the posterior entropy when all voters except one are. This is because if all votes except one are known, the last one can be inferred immediately by checking the election's results.

It should also be noted that the percentage of information leaked $I_{O,h}\%$ increases with the number of targets, again sublinearly. We argue that this is a desirable property

in a protocol designed to protect a secret composed of several subsecrets with the same importance. The property ensures that it is not more convenient for the attacker to try to infer separately every single secret instead of the whole composed secret, so even if it takes less time to infer the secret of 1 target out of 3, the time it would take to infer all 3 secrets one by one is larger than the time required to infer them all at the same time, as we explained in the paragraph above. This guarantees that the posterior entropies for multiple targets are in fact sound. This property also forces the attacker to decide beforehand exactly how many votes he needs to discover to minimize the time needed for the attack.

6 Challenges

6.1 Problem size

The examples we analyzed consider only a small number of voters and candidates. The algorithm for the precise computation of information leakage is exponential in the size of the secret and the size of the secret grows with the number of voters and candidates, thus QUAIL and any other tool are too slow to analyze large cases. Analyzing the Preference Ranking protocol also requires more time than analyzing the Single Preference protocol, since the former protocol is more complex than the latter and has a larger secret size.

To solve this problem we are implementing a statistical analyzer in QUAIL able to simulate the execution of the protocols a large number of times and to approximate the information leakage value by analyzing the collected data. The statistical analyzer is able to solve larger problems than the standard QUAIL algorithm, since it does not have to analyze the whole space of possible program executions.

6.2 g-leakage

Information leakage quantifies the loss of information on the whole secret. We have analyzed the behavior of leakage when we consider only some of the votes to be the secret we are interested in, showing how the protocol is efficient in hiding the secret of the single voters.

Recently an extension of information leakage, called *g-leakage* [3], has been proposed exactly to deal with cases in which different subsets of the secret bits have different values for the attacker, as is the case with composite secrets. g-leakage is very general, since it allows for any gain function to be used in top of leakage computation; for this reason it has not yet been implemented in any leakage analysis tool.

We are working to extend QUAIL with g-leakage computation capabilities, allowing us to encode more naturally problems with composite secrets like voting protocols. We expect that the results on the efficiency of the protocols in protecting the single votes will be coherent with the results presented in this paper.

6.3 Implementation Analysis

The protocols we analyze model the abstract behavior of voting systems, giving us theoretical lower bounds on the amount of information leaked by publishing the results

of the elections. It would be interesting to compare these theoretical results with actual implementations of voting systems, to evaluate how effective the real systems are in guaranteeing anonymity. Off-the-shelf systems are obviously not written in QUAIL language, so a tool capable of analyzing C or Java code like the ones developed by Phan and Malacaria [16] or by Chothia et al. [11] would have to be used.

References

1. QUAIL. https://project.inria.fr/quail/.
2. M. S. Alvim, M. E. Andrés, K. Chatzikokolakis, P. Degano, and C. Palamidessi. Differential privacy: On the trade-off between utility and information leakage. In G. Barthe, A. Datta, and S. Etalle, editors, *Formal Aspects in Security and Trust*, volume 7140 of *Lecture Notes in Computer Science*, pages 39–54. Springer, 2011.
3. M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring information leakage using generalized gain functions. In S. Chong, editor, *CSF*, pages 265–279, 2012.
4. G. Barthe and B. Köpf. Information-theoretic bounds for differentially private mechanisms. In *CSF*, pages 191–204. IEEE Computer Society, 2011.
5. F. Biondi, A. Legay, P. Malacaria, and A. Wasowski. Quantifying information leakage of randomized protocols. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *VMCAI*, 2013.
6. F. Biondi, A. Legay, B. F. Nielsen, and A. Wasowski. Maximizing entropy over Markov processes. In A. H. Dediu and C. Martín-Vide, editors, *LATA*, 2013.
7. F. Biondi, A. Legay, L.-M. Traonouez, and A. Wasowski. QUAIL: A quantitative security analyzer for imperative code. In Sharygina and Veith [18].
8. K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Anonymity protocols as noisy channels. *Inf. Comput.*, 206(2-4):378–401, 2008.
9. H. Chen and P. Malacaria. Quantifying maximal loss of anonymity in protocols. In W. Li, W. Susilo, U. K. Tupakula, R. Safavi-Naini, and V. Varadharajan, editors, *ASIACCS*, pages 206–217. ACM, 2009.
10. Y.-Y. Chen, J. ke Jan, and C.-L. Chen. The design of a secure anonymous internet voting system. *Computers & Security*, 23(4):330–337, 2004.
11. T. Chothia, Y. Kawamoto, and C. Novakovic. A tool for estimating information leakage. In Sharygina and Veith [18], pages 690–695.
12. M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368. IEEE Computer Society, 2008.
13. T. M. Cover and J. A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006.
14. D. Gritzali. Principles and requirements for a secure e-voting system. *Computers & Security*, 21(6):539–556.
15. P. Norris. *Electoral Engineering: Voting Rules and Political Behavior*. Cambridge Studies in Comparative Politics. Cambridge University Press, 2004.
16. Q.-S. Phan and P. Malacaria. Abstract model counting: a novel approach for quantification of information leaks. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 283–292. ACM, 2014.
17. C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27:379–423, July 1948.
18. N. Sharygina and H. Veith, editors. *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, 2013.

A Appendix

Proof (of Lemma 1). Call h the composite secret about the votes of all voters and O the observable output of the system. Remember that information leakage corresponds to the difference between the prior entropy on the secret $H(h)$ and the posterior entropy on the secret after observing the observable output of the system $H(h|O) = \sum_{o \in O} P(o)H(h|O = o)$.

The secret h has c^n possible values, thus can be encoded in $\log_2 c^n$ bits, thus the prior entropy $H(h)$ corresponds to $\log_2 c^n$.

For the posterior entropy $H(h|O)$, consider that the possible votes on the candidates form a multinomial distribution, thus the probability $P(o)$ of a given outcome $o \in O$ is $1/c^n$ and the conditional posterior entropy $H(h|O = o)$ where k_i is the amount of votes to candidate $1 \leq i \leq c$ is $\binom{n}{k_1+k_2+...+k_c}$. We conclude that

$$
\begin{aligned}
I(O, h) &= H(h) - \sum_{o \in O} P(o)H(h|O = o) \\
&= \log_2 c^n - \frac{1}{c^n} \sum_{k_1+k_2+...+k_c=n} \binom{n}{k_1 + k_2 + ... + k_c} \left(\log_2 \binom{n}{k_1 + k_2 + ... + k_c} \right)
\end{aligned}
$$

Voting Protocol Models

Single Preference The model for the Single Preference protocol is shown on Fig. 3 on the left. Constant `N` represents the number of voters and constant `C` the number of candidates. The observable variable `result` is an array with the total votes expressed for each candidate, and the secret variable `vote` is the array with the preference of each voter. The rest of the code just sums up the votes for each candidate.

Preference Ranking The model for the Preference Ranking model is shown on Fig. 3 on the right. Constant `N` represents the number of voters and constant `C` the number of candidates. The observable variable `result` is an array with the total points obtained by each candidate, and the secret variable `vote` is an array with the preference ranking of each voter. For each voter the secret has $c!$ possible different votes, corresponding to the possible complete orderings of the c candidates. The secret vote of each voter is encoded as a number from 0 to $c! - 1$, and then transformed in a preferential order with the algorithm in lines 30-57. The points for each candidate are counted by summing the points given by the preference ranking of each voter.

```
// N is the number of voters
const N:=3;
// C is the number of candidates
const C:=2;
// the result is the number of votes of each candidate
observable array [C] of int32 result;
// The secret is the preference of each voter
secret array [N] of int32 vote:=[0,C-1];
// this is just a counter
public int32 i:=0;
public int32 j:=0;
// voting
while (i<N) do
  while (j<C) do
    if (vote[i]==j) then
      assign result[j]:=result[j]+1;
    fi
    assign j:=j+1;
  od
  assign j:=0;
  assign i:=i+1;
od
return;
```

```
// N is the number of voters
const N:=3;
// C is the number of candidates
const C:=2;
// the result is the number of votes of each candidate
observable array [C] of int32 result;
// The secret is the preference of each voter, from 0 to C!-1
secret array [N] of int32 vote:=[0,C!-1];
// these bits represent the votes received by the voting machine
public array [N] of int32 decl;
public array [C] of int32 temparray;
public int32 pos;
// this is just a counter
public int32 voter:=0;
public int32 candidate:=0;
public int32 k:=0;
public int32 y:=0;
// voting
while (voter<N) do
  while (candidate<C) do
    if (vote[voter]==candidate) then
      assign decl[voter]:=candidate;
    fi
    assign candidate:=candidate+1;
  od
  assign candidate:=0;
  assign voter:=voter+1;
od
// transform the secret of each voter into the order of the preferences
assign voter:=0;
while (voter<N) do
  // build the initial array
  assign candidate:=0;
  while (candidate<C) do
    assign temparray[candidate]:=candidate;
    assign candidate:=candidate+1;
  od
assign k:=C;
  // find a position
  while (k>0) do
    assign pos := decl[voter]%k;
    assign candidate:=C-k;
    // update the vote of the candidate
    assign result[candidate]:=result[candidate]+temparray[pos];
    // remove the element from the array
    assign y:=pos;
    while (y<C-1) do
      assign temparray[y]:=temparray[y+1];
      assign y:=y+1;
    od
    // update the vote of the voter
    assign decl[voter]:=decl[voter]/k;
    // decrease the counter
    assign k:=k-1;
  od
  assign voter:=voter+1;
od
return;
```

Fig. 3: Model for the Single Preference protocol (on the left) and for the Preference Ranking protocol (on the right).